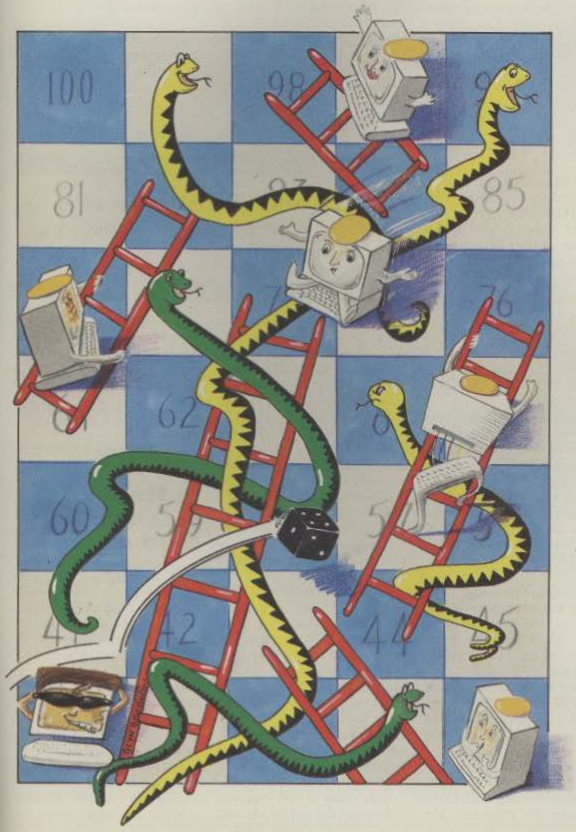
Your COMPUTER COURSE

JUNE 1986

A COMPREHENSIVE GUIDE TO PROGRAMMING – PART 4



THE ART OF ANIMATION

EXPLORING COMMODORE SPRITE MANIPULATION

SMOOTH OPERATORS

LOW DOWN ON OPERATING SYSTEMS

UNSCRAMBLING COMPUTER JARGON

UNDERSTANDING COMPUTER TALK

68xxx MACHINE CODE TECHNIQUES

MANIPULATING THE BITS AND BYTES

The halfway point in the *Your Computer Course* is now past and we hope that if you have been following the first three instalments you will by now have a firm grounding in the principles and techniques of microcomputer programming.

Now is the time to build on the knowledge gained in the first half of the course and to that end we have planned a number of more advanced yet still very understandable features to take us to part six, the last issue of the *Your Computer Course*.

Now is the time to make an apology. We jumped the gun in last month's introduction. The item on programming 68xxx processors was mentioned but did not appear in the issue. We hope it was worth the wait – you can judge by turning to page 30.

Computer graphics, a continuing theme of the course, is featured again. the star this month is the Commodore 64 and the article features a program to demonstrate some of the techniques of programming with sprites.

John Lettice's Glossary continues its unstoppable way on pages 28 and 29, while the final article this month looks at operating systems. A computer's OS can make or break the hardware of a system both in terms of performance and user-friendliness, yet many people are not sure of the role that this software plays in the overall function of a computer.

INDEX

Commodore graphics 26

Glossary (M-P) ... 28

68xxx programming 30

Operating systems . 32

Contributors:

Ken Alexander
John Lettice
Geof Wheelwright

Commodore 64 graphics

Geof Wheelwright explores the art of graphics manipulation on Commodore's 64K computer.

One of the keys to the successful writing of any computer game is the ability to move things round and across the screen. You also need, of course, to be able to control that movement using the joystick or the cursor.

Take heart, however, as you do not need to be an expert in machine code, Pascal or any specialised language to do so. A working acquaintance with the Basic provided with your machine should provide you with all the tools you need to write a simple animated and cursor-controllable game.

The place to start is the PRINT statement. To get the simplest impression of movement or animation on-screen, type-in the following lines of program and run them:

```
10 PRINT "A"
20 CLS
30 PRINT "                A"
```

The Commodore 64 does not have a CLS command, so it employs PRINT CHR$(147) to do that job.

That simple three-line program will print the letter A in the top left-hand corner of the screen, clear the screen and then print it again a few spaces to the right of where you printed it previously. As all of the instructions are carried-out in quick succession, they create the impression of screen movement.

You will not have much fun moving the letter A about the screen – you want to move aliens, camels and the other baddies which will populate your games. To do that, you first need to design them, using the USR system for user-defined graphics on the Spectrum and sprites of UDGs on the Amstrad, 64 and BBC.

Once you have defined a user-defined graphic, the job will turn to controlling that UDG with either the joystick or the cursor keys. The first thing to remember when embarking on that task is that all the cursor keys or joystick ports do is to return a value when you push the stick or press a key.

Your job is to tie the value of that keypress to the location of your UDG on the screen. The easiest way to do it on the keyboard – we will omit the joystick for now as joystick ports work in slightly different manner on different machines – is to have the cursor position, i.e., your UDG, move up when the UP key is pressed, down when the DOWN key is pressed, and so on.

Detecting which key has been pressed is done by putting the value of the character in a string variable and then checking the ASCII value of it. If, for example, that value is equal to that of the left key, you would move the UDG one to the left.

If you intend to have some baddies to combat, you will also need to design them and write a random-number generator which places them in odd places all over the screen.

The only problem remaining is what is known as collision detection, so that when your man moves over something he wants to pick up, the micro will know that he has arrived there and will do something about it. The easiest way to do it is to check the location of whatever else you have on the screen – it will normally be an X,Y value – and seeing if it is equal to the location of your on-screen man.

If the value of both locations is the same, you will have scored a hit and your score card should be increased and the image of the baddie will be blanked-out. A little more programming will allow you to check friendly beasties against enemies, so that you would bounce off them and hit enemies.

You will, of course, also want to put a score counter in the game and that is a simple matter of adding whatever value you like every time there is a hit and then displaying that somewhere on the screen. Alternatively, you just set up a simple timer which allows you to play for a set amount of time, and accomplish all your tasks within that time, as we have done in the example.

To review, you should look at the following issues to write a game:

Design a User Defined Graphic or Sprite. To move it,

The place to start at is the PRINT statement.

print the character, clear the screen, print the character with a space in front of it, clear the screen, and so on.

To add cursor control to the program, make the location of the character dependent on the value of the cursor keys you press.

Populate the screen with other randomly-located UDG baddies.

Set up a collision detection system, checking the location of baddies against your location.

Write a scoring routine and

tie it into your collision detector.

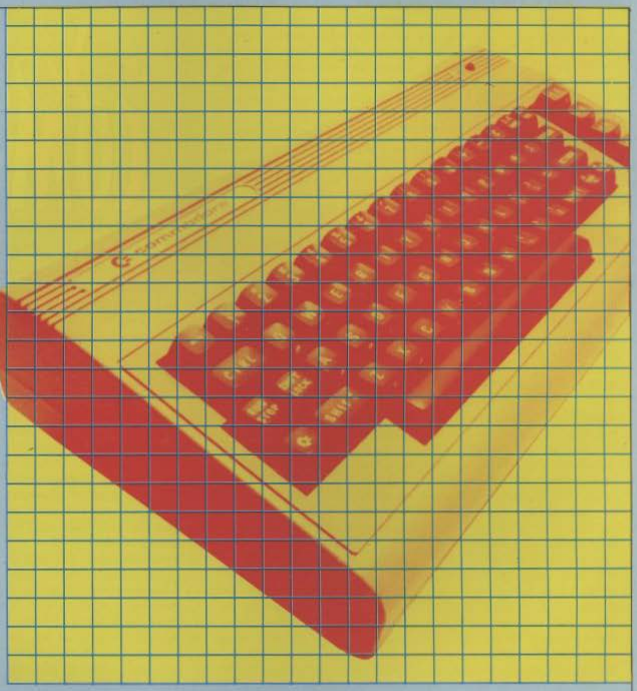
In the example program we carry-out this plan, defining the shape of a stick man on the Commodore 64, moving it across the screen and then switching into a little game where the object is to use the cursor keys to collect the little lumps of coal which populate the screen before the timer in the top left-hand corner of the screen reaches 120.

You will notice when you run the program, however, that the stick man you had originally in

The use of user defined graphics makes programming easier.

the opening screen has turned into a series of standard letters and numbers. That is intentional and designed to show how special animated characters can be made up from standard letters and numbers and to pose a challenge for you in figuring how to switch from the back into the UDGs which make up the small man.

In the next issue, we will tell you how the program can be expanded – and solve the question of switching back into UDG mode, if you have not guessed it already by then.



```
5 PRINT CHR$(147)
10 POKE 56334,PEEK(56334)AND254:POKE1,PEEK(
1)AND251
20 FOR I  0 TO 63
30 FOR J0 TO
40 POKE 12288+I*8+J,PEEK(53248+I*8+J)
50 NEXT J:NEXT I
60 POKE 1,PEEK(1)OR4: POKE56334,PEEK(56334)
OR1
70 POKE 53272,(PEEK(5322)AND240)+12
80 FOR KAR=60 TO 63
90 FOR BYTE=0 TO 7
100 READ W
120 POKE 12288+(8*KAR)+BYTE,W
140 NEXT BYTE:NEXT KAR
145 FOR L=40 TO 78
150 PRINT CHR$(14)TAB(L)CHR$(60);
160 PRINT CHR$(61)TAB(L)CHR$(62)CHR$(63)
165 FOR H=1 TO 10:NEXT H:NEXT L
170 GET A$
180 IF A$="" THEN GOTO 10
190 POKE 5322,21
200 DATA ,,,1,1,31,3,3
210 DATA 224,224,224,128,128,248,192,192
220 DATA 1,2,4,8,16,32,64,128
230 DATA 128,64,32,16,8,4,2,1
1000 PRINT CHR$(147)
1500 PRINT CH$$(5)
2000 O=1484
2100 FOR I=58296 TO 56295:POKE I,1:NEXT I
2600 FOR U=1 TO 12:X=INT(RND(1)*1000)+1024:
POKE X,35:NEXT U
3000 FOR T=1 TO 125
3100 GET B$: IF B$=""THEN 3100
4000 IF B$=CHR$(1)THEN O=O+40:POKEO-40,32:
POKEO-39,32:FMKEO,32:POKEO+1,32
5000 IF B$=CHR$(145)THEN POKEO,32:POKEO+1,3
2:POKEO+40,32:POKEO+41,32:O=O-40
6000 IF B$=CHR$(15)THEN POKEO,32:POKEO+1,3
2:POKEO+40,32:POKEO+41,32:OO-1
000 IF B$=CHR$(29)THEN POKEO,32:POKEO+1,32
:POKEO+40,32:POKEO+41,32:OO+1
8000 POKEO,60:POKEO+1,61
8200 POKEO+40,7:POKEO+41,63
9000 PRINT T:PRINT CHR$(19):PRINT CHR$(32):
NEXT T
9500 PRINT CHR$(147)
900 POKE 53272,21
10000 FOR G1 TO 23
11000 PRINT "................TIME IS UP....
........"
12000 NEXT G:GOTO 10000

READY.
```

Your COMPUTER G

John Lettice continues his guide to computer jar from machine code to programmi

■ **Machine code:** The language a computer understands, i.e., binary coding. Also known as machine language but in terms of comprehensibility code expresses the full flavour.

■ **Machine independent:** A language, program or process which is not tied to a particular machine is machine-independent. Not to be confused with a machine you cannot control, which is an independent machine.

■ **Machine instruction:** A set of randomly-generated irrelevancies, entitled Manual, and with the apparent objective of encouraging you to buy the Advanced User Guide accompanying your shiny new computer. Alternatively, a particular instruction encoded in machine code/language.

■ **Machine language:** Expletives generated by the behaviour of a given machine. Otherwise see **Machine code.**

■ **Macro:** A micro is a small computer, so computers being about logic you would expect a macro – it's Greek for big – to be a big computer – circuits like tramlines, wrought iron logic gates, that kind of thing – but you would be wrong. It is a sequence of instructions preprogrammed to be called as one unit, so if it goes "do this, do that, do the next thing", all three instructions can be called by the name of the procedure.

■ **Macro-assembler:** An assembler used to build macros.

■ **Magnetic bubble memory:** An expensive form of memory which draws little current. Now largely superseded by CMOS. See **CMOS.**

■ **Magnetic media:** Any form of storage based on the magnetic state of the medium used. Also specialised newspapers dealing with magnetism, most famous being the now-defunct *Magnet.*

■ **Mainframe:** A large but practically indefinable computer, as micros trespass on the territory of minis and minis similarly poach on mainframe territory. Originally mainframes were large, powerful computers, but they are now best defined as any large computer where the gubbins is contained in a steel frame.

■ **Main memory:** The part of a computer memory which can be accessed directly by the main processor, as opposed to external storage or memory configured to behave like external storage.

■ **Mantissa:** A kind of Spanish dance. Also the part of a number to the right of the decimal point.

■ **Mask:** A piece of hardware used by designers of failed computers for protective purposes. Also a method used to prevent the CPU being interrupted by an external device.

■ **Mass storage:** An external storage device, usually a hard disc, but could be a floppy disc, depending on your concept of storage. True mass storage is, or course, a hard disc used by a priest.

■ **Maths:** A discipline thought by the ignorant to be endangered by technology but those in the know realise it is even more important nowadays. See **Floating point arithmetic.**

■ **Matrix:** A table of numbers. Multiplication tables are an example of a two-dimensional array but they can also be of three, four or whatever dimensions. Many programs fail to work because their arrays are of "whatever" dimensions.

■ **Matrix printer:** See **dot matrix printer.**

■ **Megabyte:** One million bytes, which is, in today's terms, a grossly inadequate amount of memory. Business computers used to operate satisfactorily with 64K but now they need 512K just to generate a syntax error.

■ **Memory:** Disc storage is also technically memory but the term generally is used of the volatile memory on board a computer.

■ **Memory bank:** A bank of memory connected to the CPU in tandem with one or more other banks, allowing the CPU to switch between them.

■ **Memory dump:** The computer graveyard, or a printout of the contents of a computer memory, usually used in an emergency where the contents of memory must be reconstructed later.

■ **Memory map:** A chart issued by the manufacturer to confuse you about the functions of the various parts of a computer memory.

■ **Menu:** A series of options displayed on-screen by a program, e.g., 1, Edit; 2, Print; 3, Cod and chips.

■ **Menu-driven:** Meals on wheels or a kind of program which allows you to control it by moving from one list of options to another.

■ **Microcomputer:** Literally a very small computer but in reality a computer so compact and inexpensive that almost everyone can be annoyed by it.

■ **Micron:** A unit of length which equals one-thousandth of a millimetre. Based on the Greek-derived prefix micro, meaning small, it is apparently a very small 'n'.

■ **Minicomputer:** A small mainframe, a large microcomputer, or something in between . . .

■ **Mips:** A measure of computer speed, in millions of instructions per second.

■ **Mnemonic:** An abbreviation selected to help you remember something and used in assembly language programming to help define what a given instruction is doing.

■ **Mode:** Some computers can be switched between, say, graphics mode, text mode and so on.

■ **Modem:** Short for modulator/demodulator, a device used to convert a computer signal to and from an audio signal for transmission on a telephone line.

■ **Modular:** A way of building a system whereby blocks can be slotted in at a later date. Used by manufacturers to allow them to supply parts late.

■ **Monitor:** A kind of program used to oversee the functions of a computer, or a TV-like display used with a computer.

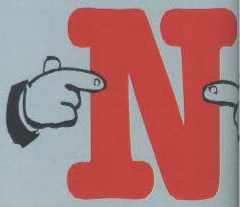
■ **Most significant bit:** The bit the manufacturer omitted from the box, or the left-most bit in a binary number.

■ **Motherboard:** A breadboard is something you use on which to cut your bread, so a motherboard is the main circuit board of a computer. Well, what did you think it was?

■ **Multi-dimensional array:** An array of more than one dimension and generally of one more dimension than that of which you can easily keep track.

■ **Multiplexor:** An item of body-building kit, or a device which allows two or more signals to share the same wire. Sometimes abbreviated to MUX.

■ **Multi-tasking:** A kind of computer which can perform more than one task at the same time. The ability to have a funny flickering on the screen and a smell of burning emerging from the main unit does not count as multi-tasking.



OURSE

n – this month he moves language.

■ **Nanosecond:** One-thousandth of a microsecond, or one-billionth of a second. Named in honour of Sir Herbert Nano, inventor of the cuckoo clock.

■ **Nesting:** Having one or more routines inside another, e.g., nested FOR . . .NEXT loops. A technique used for storage of parts in Sir Herbert Nano's first factory.

■ **Network:** A group of computers linked through communications. Because of the difficulties involved in connecting a working network, the similarity to "notwork" is not necessarily coincidental.

■ **Nibble:** Four bits, or half a byte.

■ **Noise:** Interference on a – generally communications – line. Generally less obvious than "'Ere, what's that 'orrible shrieking noise?"

■ **Non-maskable interrupt:** An interrupt which cannot be masked. See **mask.**

■ **Non-volatile:** Obsolete term meaning a British micro company which is not going out of business, or a storage device which retains its data even when the power is shut off.

■ **Null:** Zero, or an electrical signal defined as zero. Now you know what "Norway, null points" means.

■ **Null string:** A string which contains no characters.

■ **Number-cruncher:** A term used of a large computer used predominantly to perform large numbers of calculations and to send bills for 0.00 pence.

■ **Numeric:** A number, or an operator such as +, –, used with numbers.

■ **Object code:** The code output of an assembler or compiler.

■ **Obsolescence:** From the manufacturer's point of view, the most important component in your computer.

■ **Octal:** A numbering system using the base eight. Used by heavy programmers as shorthand for binary but otherwise of no use to man nor beast.

■ **Off-line:** Not connected – a micro which is not connected to a mainframe is off-line.

■ **Ones complement:** A binary number produced by reversing the ones and zeros in the original number.

■ **On-line:** Two devices connected to one another are on-line.

■ **Opcode:** Often used as an alternative to machine code – short for operation code.

■ **Operand:** A word or number operated on logically or arithmetically.

■ **Operating system:** A group of programs put together to give a computer its personality and allow it to work. It is not clear why the latter part seldom becomes true. Commonly abbreviated to O/S.

■ **Operator:** A mathematical or logical symbol, e.g., +, used to represent an operation between variables or constants.

■ **Optical character recognition:** The identification of characters by a scanner connected to a computer. If computers could read properly, there would be no need for filing cabinets. Obviously they can't – yet.

■ **Output:** Information produced by a computer and sent to an external device. See **Error message.**

■ **Overflow:** What happens when a calculation exceeds the size of the register containing it. If you imagine dirty bathwater slopping around inside your computer you will get completely the wrong idea.

■ **Overlay:** A method of programming which uses the same area of memory at different times by over-writing parts of the program no longer needed. That is why flasher applications look as if they are dying at regular intervals – they are re-writing themselves.

■ **Package:** Term used by the software industry to describe a program, no doubt deriving from the most costly part of the finished product.

■ **Packet:** A batch of transmitted data.

■ **Paged memory:** A method used to switch areas of memory in an out of the CPU address space.

■ **Pagination:** The technique whereby continuous stationery of 72 lines per sheet becomes 73 lines per sheet when you set your word processor for 72.

■ **Palindrome:** A binary number which reads the same from left to right and from right to left.

■ **Parallel:** A kind of interface where the signals move side by side, or in parallel, rather than in-line, in serial.

■ **Parallel processing:** A computer system capable of doing two or more things simultaneously is capable of parallel processing.

■ **Parameter:** A variable entered from one part of a program into another.

■ **Parity:** A method of checking for error in data communications, where parity is set either at odd or even.

■ **Parse:** To separate a string into its parts before attempting to understand it. Parsers are used in adventure programs to simulate understanding on the part of the computer.

■ **Pascal:** A high-level language much favoured by the Campaign for Real Programming (CARP). See **Techie.**

■ **Pass:** A single run through a program or subroutine.

■ **Password:** Often used in networks or multi-user systems to protect the individual's data. Passwords are in two categories – easy-to-remember ones which anyone can guess and difficult ones you can never remember. There is no middle ground.

■ **Patch:** A small amendment to software "patched-in" to a main program to correct errors.

■ **PCB:** Printed circuit board.

■ **PEEK:** Basic command used to return the value present at the address PEEKed.

■ **Peripheral:** An extra which is controlled by, but not part of, a computer.

■ **Pin-out:** The layout of the connections in a circuit or plug.

■ **Pixel:** Picture element, the smallest component of a computer display.

■ **Plotter:** Member of a group of manufacturers trying to carve up IBM, or an output device which uses pens to draw graphs and diagrams. Cheaper variants sometimes double as printers.

■ **POKE:** Basic statement used to POKE a value directly into a memory location.

■ **Port:** A connector used for I/O to printers and the like.

■ **Portability:** Imaginative term used by software manufacturers who claim their software can be transferred easily from one machine to another.

■ **Printhead:** The part of the printer which transfers the image to paper.

■ **Printout:** The hard copy produced by a printer.

■ **Procedure:** Often used interchangeably with subroutine.

■ **Program:** Any list of commands which can be stored – even if only in RAM – is a program. It need not, and often does not, work.

■ **Programmer:** In Darwinian terms, somewhere on the evolutionary scale between artichoke and three-toed sloth.

■ **Programming language:** A set of terms understood by the computer and programmer to have a common meaning.

First steps with 68xxx machine code

David Janda introduces the basics of 68xxx machine code programming which in many respects is more straightforward than eight bit coding.

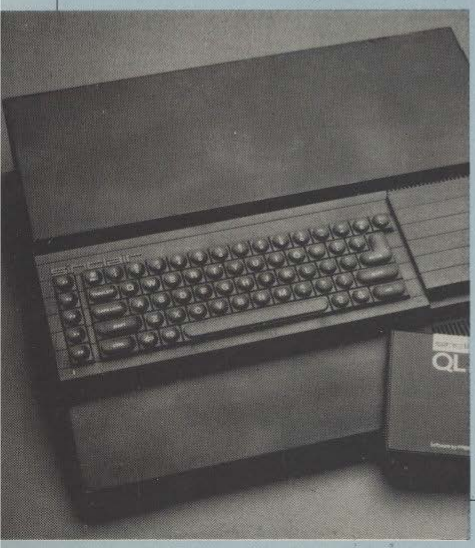
Many people are deterred from the idea of learning machine code because it takes a great deal of effort to do even the simplest things. Take a simple task such as multiplying two small numbers; it can be a real problem for the beginner in machine code and it is understandable when learners are frustrated.

The reason even simple tasks take up much time to code is because of the way the central processing unit is designed. Most home micros use one of two CPUs, the 6502 or Z-80. They are 8-bit CPUs, meaning that the CPU registers can hold values up to eight bits in length. That is 11111111 in binary which is 256 in decimal. The CPU native instructions are also very simple; operations such as add, subtract and load are about as complex as you will get in an 8-bit machine. Z-80 programmers are lucky insofar as they can double-up registers, thus permitting them to be used to store larger values. So register B and C can be regarded as a register pair BC. That will allow BC to store a 16-bit value which is 1111111111111111 or 65536 in decimal. Even that is not good enough for some applications.

As more and more computers are made, the cost of developing components falls. That means that more and more memory is available at cheaper prices. That is also true for CPUs. The 16-bit CPU which is used in the business end of the market extensively is now becoming available to the home user at realistic prices.

There are two machines available to home users which use the 16-bit 68000 CPU, the Sinclair QL and the Atari 520 ST. Those machines are more powerful in processing terms than anything previously available and that can be attributed to the design of the 68000. The 68000 is a 16/32-bit CPU which was designed by Motorola. There are four chips in the 68xxx family – 68000 as used in the Atari ST range; 68008 as used in the Sinclair QL; 68010; and 68020.

The difference between them is the size of the data bus which is used to fetch and store data to and from memory. All have an internal 32-bit architecture but in the case of the Atari ST the data bus is only 16 bits wide and on the




```
The eight data registers

31                                  0
 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX D0

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX D1

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX D2

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX D3

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX D4

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX D5

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX D6

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX D7
```

Sinclair QL eight bits wide. In practice, that means the 68008 running the same program as the 68000 would do so, but somewhat slower.

Unlike its 8-bit counterparts, the 68xxx series has plenty of registers – 19, all except one 32 bits long. The first set of registers are eight data registers which can be used like variables. Having eight data registers is useful as it permits a programmer to jiggle about with data without having temporarily to store it in main memory, which is time-consuming. Those registers may be used for arithmetic and logical operations. Four types of data can be used with this type of register – individual bits, 8-bit bytes, 16-bit words and 32-bit long words. That means, for example, that a program could work with four 8-bit characters held in one data register at the same time.

The 16-bit CPU is now widely available to the home user at realistic price levels.

The data registers can also be used to manipulate whole signed integers within the range +2,147,483,647 and –2,147,483,648. Because programmers will not use the full 32 bits, the majority of arithmetic operations can work only on part of a data register. A byte operation uses only bits zero to seven of the data register. A word operation uses bits zero to 15 and some arithmetic operations can use all 32 bits.

The next set of registers to consider is the address registers. They are used mainly to point to areas of memory where blocks of data are stored, such as arrays, character strings and so on. There are, however, some arithmetic operations which can be performed on this set. Register A7 is rather special insofar as it holds the stack pointer value. There are two A7s, one used for the user stack pointer and the other the system stack pointer. That is because the 68xxx series can run in two states, system and user.

The final two registers are the Program Counter and the Status Register. The PC is used to keep track of the address of the next instruction to be executed. The SR records the status of certain aspects of the last operation performed. That information can be used to decide whether to jump to a subroutine, exit a subroutine and so on.

The 68xxx series has about 57 basic instruction. Instructions which are found on many 8-bit CPUs are included – Addition – ADD; subtraction – SUB; jumps, moves, branches and bit manipulation. What is not common to 8-bit micros are instructions for multiplication, MUL; and division, DIV; and, of course, MOVE. The ability to step through a program one instruction at a time is standard with the 68xxx series.

68xxx code can be more compact than eight-bit programs.

The MOVE instruction is very versatile and can be used to move data to and from memory or registers. It can be used in the form MOVE.W which means that the data to be moved is a long word – 32 bits. MOVE.W on the other hand means that the data to be moved is 16 bits long. Here is an example:

MOVE.W $21232,D1

Move the 16-bit value stored at address $21232 into data register D1

That is absolute addressing; $21232 is not an offset of another address – it means address $21232.

To perform the opposite, that is store the contents of D1 into an address, the following could be used:

MOVE.W D1,$32767

Move the 16-bit word stored in D1 into the address at $32767.

Addition is equally simple. ADD.W $26352,D3 would add the contents of address $26352 to that already stored in register D3. SUB and MULT can be used in a similar manner, and immediate addressing is just as easy to grasp, the point being that with the 68xxx series it is feasible to write a program in machine code which performs a mathematical task which would need much more code on an 8-bit machine.

```
The nine address registers

31                                  0
 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX A0

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX A1

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX A2

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX A3

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX A4

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX A5

 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX A6
       user stack pointer
 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX A
 XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX 7
     supervisor stack pointer


END
```

Operating Systems – a key role

Behind every great micro is a good operating system. Ken Alexander explains the importance of a computer's OS.

Every schoolboy knows the difference between computer hardware and software; hardware hurts if you drop it on your foot, software does not. Where does firmware fit into this simple view of the computer? Well, it could be said that firmware would hurt if dropped on certain parts of your anatomy – but not much.

Joking aside, though, firmware, in the form of a resident operating system, plays a vital part in the function of any computer. The *Your Computer Course* has in recent months explained how computers may be programmed both in high-level, interpreted languages and in low-level machine code, the native tongue of the microprocessor at the heart of the computer. Whether programming in high- or low-level languages, there is another essential program working away in the background – the computer operating system.

The major function of the operating system is to take care of what are termed housekeeping functions in the jargon of computing. Despite John Lettice's entry in the course glossary, housekeeping does not mean that computers do the washing before they go to work – at least not yet.

Housekeeping tasks include such operations as collecting data from the computer keyboard, displaying output on the system VDU, controlling the operation of any serial or parallel input/output ports and taking care of the saving and loading of data to a mass storage device, be it a cassette recorder or disc drive. When programming in a high-level language, all those operations are taken for granted and, even at machine code level, in general programmers will pass data to a routine within the OS when they wish to do so – for example, input or output data from a program.

When designing a computer system, manufacturers have an important choice. Either they can write an in-house OS or they can opt for an existing system and tailor it to their machines. Many home computers of an early 80s vintage have their own OS; such machines include the Spectrum, CBM 64 and BBC micro.

The disadvantage with manufacturers' own-brew operating systems is that a program written for any one machine can be used only with that particular computer – the software is not portable. The trend in more recent times, though, is for an increasing number of eight home computers to adopt one of the best-known operating systems, CP/M. CP/M software is portable thus, at least in theory, a CP/M program written for, say, the Amstrad CPC6128 will be able to run on the Commodore C128, as both machines support CP/M.

The trend in future will be for every micro to support at least one industry standard OS – CP/M for eight-bit machines, Gem or PC-DOS for 16-bit designs.

Given that the architecture of any two computers can be very different, how does an OS manage to achieve the portability of software? The secret is in the fact that any operating system consists of a kernel program which manages the system memory and the way it interacts with the processor. Any computer which supports a particular OS must conform to given standards in this area. It is the way in which I/O is handled which gives the OS its portability.

The major function of an operating system is housekeeping.

Essentially, an OS adopts what is generally termed a device-independent approach to input and output from a system. That means that the operating system will treat all I/O to a particular class of device – VDU, keyboard or disc drive – in an idealised way. That takes no account of the particular requirements of the device in a 'real' computer. The trick is that any manufacturer wishing to use the OS on its hardware writes a small piece of code which translates the flow of data to and from the OS's ideal I/O driver and translates it into a form which can be accepted by the physical device on its computer.

Installing an OS, then, given that the design of the memory and processor meets certain standards, is simply a matter of writing a series of I/O drivers.

The trend in future will undoubtedly be that all computers will support at least one industry-standard OS. For 8-bit micros that will be CP/M; in the case of 16-bit hardware the choice of OS will depend on the computer CPU. 68000 hardware will adopt Gem while PC-DOS is the standard for machines built round the Intel series of MPUs.

Soft systems

While in the main article it is assumed that a computer operating system is supplied as part of its firmware, there is an increasing trend towards supplying the OS as disc-based software. The leading example of that approach are the Amstrad PCW computers.

The CP/M Plus operating system which forms part of the bundled software supplied with those computers is provided on a 3in. floppy disc – the computer contains no resident operating system. If there is no OS, the software which controls all a machine's I/O, how then does the PCW computer manage to laod CP/M from disc? The answer lies in the fact that there is a small program resident in the machine, although it could scarcely be described as an operating system.

It is a bootstrap routine, its job being to load the first few bytes of data from the computer disc drive – that code is the core of the operating system and soon takes over control of the system from the bootstrap routine.

The supply of a machine's OS as soft, rather than firmware, has a number of advantages to manufacturer and user alike, the main advantage being that it is a simple matter to issue upgraded versions of the software. It has the disadvantage that the OS occupies an amount of computer RAM, although as the memory size of each new generation of computers continues to increase, that is no longer a serious handicap.